A Coordinated Plan for Teaching Software Engineering in the Rey Juan Carlos University

Jorge Enrique Pérez-Martínez, Almudena Sierra-Alonso
Rey Juan Carlos University
C/Tulipán s/n, 28933
Móstoles (Madrid)- Spain
{j.perez, a.sierra}@escet.urjc.es

Abstract

Nowadays both industry and academic environments are showing a lot of interest in the Software Engineering discipline. Therefore, it is a challenge for universities to provide students with appropriate training in this area, preparing them for their future professional practice. There are many difficulties to provide that training. The outstanding ones are: the Software Engineering area is too broad and class hours are scarce; the discipline requires a high level of abstraction; it is difficult to reproduce real world situations in the classroom to provide a practical learning environment; the number of students per professor is very high (at least in Spain); companies develop software with a maturity level rarely over level 2 of the CMM for Software (again, at least in Spain) as opposed to what is taught at the University. Besides, there are different levels and study plans, making more difficult to structure the contents to teach in each term and degree. In this paper we present a plan for teaching Software Engineering trying to overcome some of the difficulties above.

1. Introduction

In Spain there are three Computer Science degrees: Technical Engineer in Software Management (TESM) (this degree specializes in Information Systems), Technical Engineer in Systems Computing (TESC) and Computer Engineer (CE). The two first degrees are given after a first level (somewhat similar in contents to the bachelor degree) with a three year duration. Students with one of these degrees can continue studies in CE (similar in contents to Master studies) with a duration of two years (see Figure 1). The Ministry of Education establishes by law [4] the minimum number of credits of a set of disciplines considered essential and mandatory to obtain these degrees. One credit is equivalent to 10 class hours. There are theory credits and practical (laboratory) credits. Disciplines are organized in different courses and academic years. Afterwards, each university, taking into account the mandatory disciplines, defines its study plan orienting and specializing such studies. Specialization is accomplished by increasing the number of credits of certain mandatory subjects and/or including elective courses. Consequently, in Spain there is a common set of courses for all students in the same degree, with a high diversity of study plans (one per university).

In Table 1 we show the credits established by the Ministry of Education for the Software Engineering (SE) discipline in the three degrees.

The Rey Juan Carlos University offers the three degrees described above. In Table 2 we show the credits established by this university to obtain the three degrees in Computer Science. Mandatory and elective credits are specified in this table. The credit load for SE, divided into theory and practical credits, varies significantly from one degree to another (see Table 3). For example, this table shows that the number of credits assigned to SE for the TESM degree is almost three times the number assigned for the TESC degree. On the other hand, students arriving to the CE degree may have different levels of training, since they can come from TESM studies or from TESC studies. To this, we can add the fact that in the first year of the CE studies we can have students coming, not only from our university, but from any other university in Spain, which implies a high diversity in levels of training because of differences among study plans.

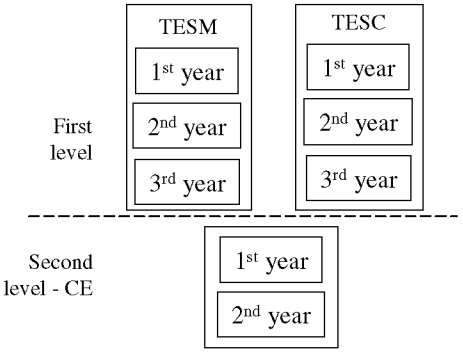

Figure 1. Organization in academic years of degrees TESM, TESC and CE.

Degree	Mandatory credits
Technical Engineer in Software Management	12
Technical Engineer in Systems Computing	0
Computer Engineer	18

Table 1. Mandatory credits for SE in Computer Science degrees in Spain.

Finally, we have to point out that the SE discipline is taught in several courses in the three degrees and it is necessary to coordinate the contents included in each of them in order to assure that the SE training obtained by the students is complete and non-redundant. In Figure 2 we show those SE courses. The courses defined as mandatory by the Ministry of Education or by the study plan in the University are marked as (M). The elective courses are marked as (E). Following the name of the course, we show the number of credits: theory credits + practical credits. We also show whether the course is taught in the first or second semester of the year. Observe that some courses extend over the two semesters.

	Credits to obtain a degree		
	TESM	TESC	CE
Mandatory	180	180	93
Electives	45	45	54
Total	225	225	147

Table 2. Credits in Computer Science degrees in the Rey Juan Carlos University.

Degree	SE mandatory credits
Technical Engineer in Software Management	6 theory + 6 practical
Technical Engineer in Systems Computing	3 theory + 1.5 practical
Computer Engineer	10.5 theory + 7.5 practical

Table 3. Credits established for SE in the Rey Juan Carlos University study plan.

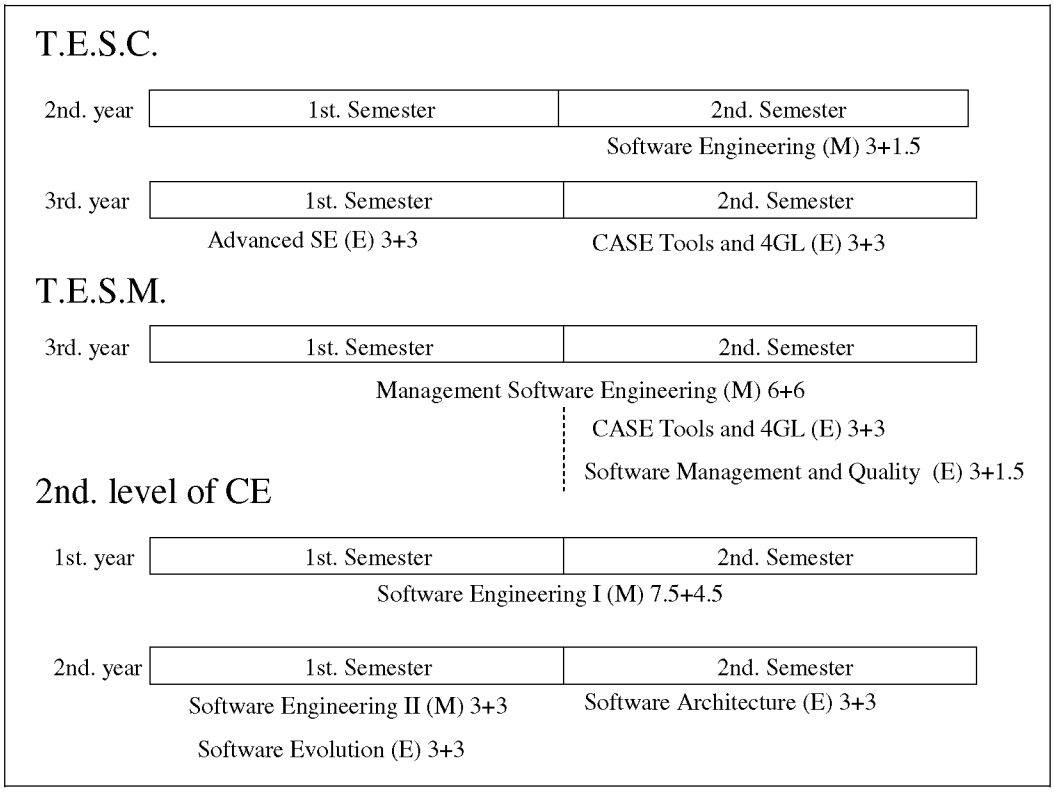

Figure 2. Distribution of SE courses by degrees, years and semesters.

Some elective courses present in the study plan have not been offered yet. That is the case with the courses *Software Management and Quality* and *Software Evolution*.

Summarizing the information presented, Computer Science studies are structured in two levels, and students may end their studies after the first one or continue studying CE. In this framework, our paper tries to solve the difficulties arising from the training differences among students of SE when they access the CE degree studies. At the same time, it tries to coordinate the set of courses in the SE area to provide a consistent view of this discipline along the different periods in which it is taught. More specifically, our goals are:

1. To give students who finish the first level (in TESM or TESC) a wide knowledge of areas related to SE as described in SWEBOK [5] and in the Computing Curricula 2001 [2] (pp. 147-153). The IEEE SE curriculum [3] has not been considered because

it is only a draft at the moment. In the second level we will deepen in this knowledge and will extend it with specialized techniques of SE and with the last advances in the discipline.

2. To establish mechanisms to provide students that access the second level from any first level degree with a similar body of knowledge in SE.

Besides, this work takes into account other problems like the extent of the Software Engineering area, the scarce number of class hours, the high level of abstraction required by this discipline, the difficulty to reproduce real world situations in the classroom, the high number of students per professor (in Spain) and the reality of companies developing software rarely over level 2 of the Capability Maturity Model for Software (again, in Spain).

The rest of the paper is structured as follows. In sections 2 and 3 we present a plan oriented to achieve the previous goals. In section 4 we show some relationships among the courses of SE. Section 5 describes some results obtained since the start of this plan. In section 6 we present the main conclusions of this work, and finally, in section 7 we outline some future work for the improvement of the proposed plan.

2. Planning for the TESC and TESM degrees

Figure 2 shows that there is an unbalance in credits between the degrees TESC and TESM in relation with the study of SE. The mandatory course has 12 credits assigned in TESM and only 4.5 credits in TESC. The only way to palliate this difference is to recommend students in the TESC studies who plan to continue with the second level to take the elective course *Advanced Software Engineering*. This way, in the TESC degree students take 10.5 credits of SE as opposed to 12 taken in the TESM degree.

Given that students in the TESC studies are free to choose or not the course *Advanced Software Engineering*, the teaching of SE must be structured in such a way that the basic knowledge of SE is assigned to the second year, and more specialized knowledge is assigned to the third year. With this idea in mind, the course *Management Software Engineering* of the TESM degree (focused on SE techniques for information systems) can be structured in two parts: the first one corresponding in contents with a deeper treatment of the course *Software Engineering* of the TESC studies and the second one with the contents included in the course *Advanced Software Engineering* of the TESC degree (see Figure 3).

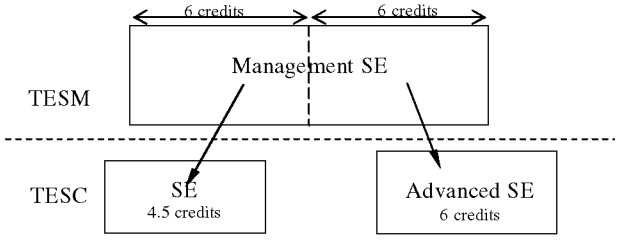

Figure 3. Relationships among the SE courses in the TESM and TESC studies.

In the way just described, we satisfy the two goals in our introduction, that is, to teach the basic concepts of the discipline and to provide students in both degrees with a similar body of knowledge in SE. The SE courses in both degrees have been developed following this plan. The contents covered in these courses are shown in Table 4. This table also includes the description of the type of practical works to develop and the tools used for them.

Tech. Engineer in Software Management	Technical Engineer in Systems Computing
MANAGEMENT SOFTWARE ENGINEERING **Goals**: to provide the student with the basic principles of Software Engineering as a framework for the construction of reliable and maintainable software. It studies current methods, techniques and tools provided for Software Engineering, and provides students with a global view of the different activities and variables to consider in the software development process. **Contents** SOFTWARE ENGINEERING Introduction to software engineering Software development models Software development methodologies O.O. SOFTWARE DEVELOPMENT Object oriented basic concepts Requirement engineering Object oriented analysis Object oriented design Implementation Object oriented software testing STRUCTURED SOFTWARE DEVELOPMENT Structured analysis Structured design Structured software testing SOFTWARE DEVELOPMENT MANAGEMENT Introduction to management tasks Software project management Configuration management SOFTWARE MAINTENANCE Introduction to software maintenance **Laboratory work** To do the O.O. analysis and design of a small application (4 or 5 use cases) from its specification, using the Unified Software Development Process. To do the time and resource planning for a software development problem. **Tools**: Rational Rose and Microsoft Project.	**SOFTWARE ENGINEERING** **Goals**: to introduce the student in the knowledge of the basic principles of Software Engineering. It studies the basic tasks of software development from an object oriented point of view. **Contents** SOFTWARE ENGINEERING Introduction to software engineering Software development models Software development methodologies O.O. SOFTWARE DEVELOPMENT Object Oriented Basic Concepts Requirement engineering Object oriented analysis Object oriented design Implementation Object oriented software testing SOFTWARE MAINTENANCE Introduction to software maintenance **Laboratory work** To do the O.O. analysis and design of a small application (1 or 2 use cases) from its specification, using the Unified Software Development Process. **Tools**: Rational Rose. **ADVANCED SOFTW. ENGINEERING** **Goals**: to learn the basic aspects of software development management and to apply methods and techniques of the structured development paradigm. **Contents** SOFTWARE DEVELOPMENT MANAGEMENT Introduction to management tasks Software project management Configuration management STRUCTURED SOFTWARE DEVELOPMENT Structured analysis Structured design Structured software testing **Laboratory work** To do the time and resource planning for a software development problem. **Tools**: Microsoft Project.

Table 4. Contents of the SE courses for the TESM and TESC degrees.

The elaboration of the contents is based on the topics described in the SWEBOK. In Figure 4 we enclose with dotted lines the topics of the SWEBOK treated in the TESM studies. The figure shows that students must have a global view of development activities, and must learn some management aspects and the basic concepts of software maintenance. Evidently, not all topics are studied with the same wideness and depth described in the SWEBOK. The topics studied with more depth are the object oriented requirements, analysis, design and testing. Students will do some practical work about these topics.

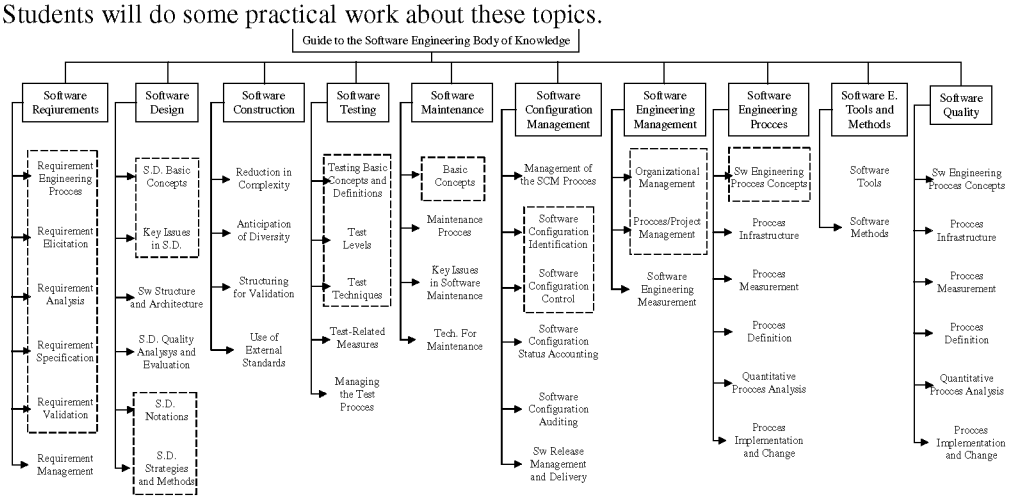

Figure 4. Contents of the SWEBOK included in the TESM studies.

In Figure 5 we show the topics treated in the TESC degree. With a uniform dotted line we enclose the topics of the mandatory course. With a non-uniform dotted line and rounded corners we enclose the topics covered in the elective course (*Advanced Software Engineering*). This figure shows that adding the two courses, we cover the same topics, but with a different depth in some of them due to the difference in the number of credits. Students that take only the mandatory credits know at least the basic development activities using the OO paradigm.

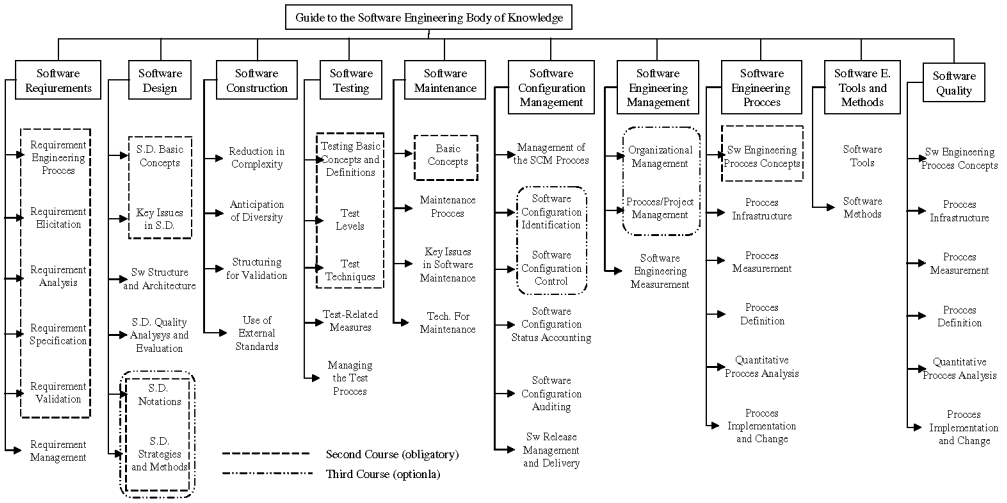

Figure 5. Contents of the SWEBOK included in the TESC studies.

Technical Engineer in Software Management and Technical Engineer in Systems Computing
CASE TOOLS AND 4GL **Goals:** To learn the importance of the CASE tools in the lifecycle of a software product, both in the development and maintenance stages. **Contents** INTRODUCTION COMPONENTS OF THE CASE SYSTEM: TOOLS SOFTWARE DEVELOPMENT SUPPORT TOPICS RELATED TO THE CASE TECHNOLOGY **Laboratory work** To develop the use case model, analysis and design of a given application. **Tools:** Rational Rose

Table 5. Contents of the course *CASE Tools and 4GL* taught in the TESM and TESC degrees.

Finally, there is an elective course, *Software Management and Quality*, that has not been opened yet. This course will make more emphasis on quality aspects because software management has already been introduced in mandatory courses.

3. Planning for the CE degree

To plan the courses of SE corresponding to the CE degree we assume that students already have a wide knowledge of all the activities in the discipline, and that they have done some practical work in development (in particular with the Unified Software Development Process).

The contents proposed for the course *Software Engineering I* of the first year, study with some depth some activities related to the discipline (with less emphasis in implementation activities for which students are supposed to have the required ability). The table of contents for this course is shown in the left column of Table 6.

For the course *Software Engineering II* of the second year, we propose the table of contents shown in the right column of Table 6.

As in the previous case, the elaboration of these contents is based on the topics described in the SWEBOK for the CE degree. In Figure 6 we enclose with a dotted line the topics of the mandatory course of the first year. With non-uniform dotted line and rounded corners we mark the topics covered in the mandatory course of the second year. As can be seen, some of the topics marked in this figure were already covered in the first level degrees. The difference is that in the second level, these topics are treated with more depth and more complex techniques are studied.

Furthermore, in the course of the second year newer subjects are treated not yet included in any of the knowledge areas of the SWEBOK. Such is the case with components or reengineering, topics included in appendix D of the SWEBOK.

In the course *Software Architecture* the student learns the last advances in this discipline that, despite its youngness, is starting to consolidate some principles and to stand out like one of the key pieces in software construction and maintenance. The contents of this course (see Table 7) correspond partially to the topic *Software Structure and Architecture* of the

knowledge area *Software Design* and partially to contents treated in appendix D of the SWEBOK.

Computer Engineer	
SOFTWARE ENGINEERING I. **Goals**: to study in depth the main tasks done in the software engineering discipline: requirements, architecture, low level design, verification and validation, project management, software quality and reengineering. The student must be able to work in any of the stages of the software development and maintenance, from the management and development points of view. **Contents** INTRODUCTION: SOFTWARE PROCESS SOFTW. DEVELOPMENT MANAGEMENT REQUIREMENT ENGINEERING SOFTWARE ARCHITECTURES DESIGN PATTERNS VERIFICATION AND VALIDATION SOFTWARE QUALITY REENGINEERING **Laboratory work** To do the planning and management of the project (configuration management, planning adjustments). To elaborate a requirement specification according to the standard IEEE 830. To develop an application from the specification, using patterns, defining the architecture, … To do the unitary and functional testing. **Tools**: RequisitePro, Rational Rose and Rational testing tools.	**SOFTWARE ENGINEERING II.** **Goals**: • To understand what a software process is and the problems to design it, and to familiarize with the metaprocess level of the Unified Software Development Process. • Familiarize students with specialized techniques of SE in environments like critical systems or formal specification. • To learn the last advances in SE like WEB engineering, component engineering or extreme programming. **Contents** SOFTWARE DEVELOPMENT METAPROCESSES DISTRIBUTED OBJECTS ENG. COMPONENT BASED SE CRITICAL SYSTEMS WEB ENGINEERING FORMAL METHODS "EXTREME PROGRAMMING" **Laboratory work** It varies every year, but it is always related to theoretical contents. **Tools**: Rational, free distribution middleware, .NET.

Table 6. Contents for the mandatory SE courses in the CE degree.

4. Relationships among the courses

The plan proposed has been implanted in the second and third year of the TESM and TESC studies. During the academic year 2002-2003, it will be implanted in the course *Software Engineering I* (first year of the second level of CE) and during the next academic year in the courses *Software Engineering II*, *Software Architecture* and *Software Evolution* (second year of the second level of CE) completing the implantation of the whole plan. Up to now, the new plan implanted in the first level has existed side by side with the previous one in the second level of CE. Because of that, the relationships among the courses shown in Figure 2 are the following:

- The course *CASE Tools and 4GL* shows the use of the tool Rational Suite Enterprise Edition 2001A. This tool is used in the practical work of the courses *Management Software Engineering*, *Software Engineering I*, *Software Engineering II* and *Software*

Architecture.

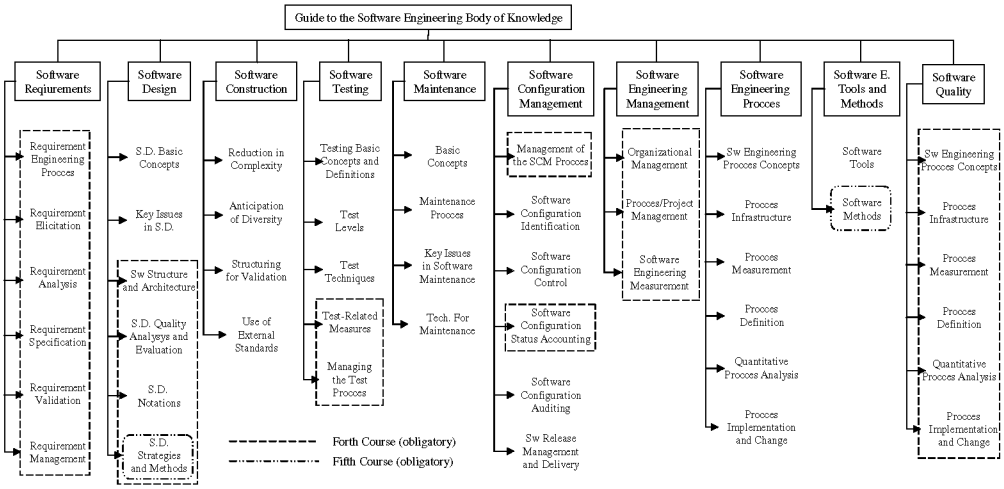

Figure 6. Topics of the SWEBOK included in the second level of the CE degree.

Computer Engineer
SOFTWARE ARCHITECTURE. **Goals**: • To recognize the architectonical styles existing in software systems. • To estimate reasonable architectures for an application and choose among them. • To be able to reason about the properties of different style architectures. • To understand how to use the domain knowledge to specialize an architecture for a particular family of applications. **Contents** INTRODUCTION ARCHITECTONIC STYLES STUDY CASES TOWARDS A SOFTWARE ARCHITECTURE DOMAIN SPECIFIC ARCHITECTURES SOFTWARE CONNECTORS ARCHITECTURE DESCRIPTION LANGUAGES SOFTWARE ARCHITECTURE DINAMISM THE ROLE OF UML IN SOFTWARE ARCHITECTURES FROM THE ARCHITECTURE TO THE IMPLEMENTATION QUALITY ATTRIBUTES OF AN ARCHITECTURE SOFTWARE ARCHITECTURES FOR USER INTERFACES **Laboratory work** Architecture analysis and design **Tools:** free distribution (ArchStudio 3.0, ACME, ...)

Table 7. Contents for the course Software Architecture of the CE degree.

- The practical work done by students of the course *Software Engineering II* (consisting

on the elaboration of a software requirements document) is used as the specification for the practical work to do by students of the course *Management Software Engineering* of the TESM degree.

- The practical work done during the first semester in the course *Management Software Engineering* is used as the specification for one of the practical works of the course *CASE Tools and 4GL.*
- The practical work done in the course *Software Engineering II* constitutes the specification for the practical work to develop in the course *Software Architecture.*

Our goal is to increase the number of relationships in the direction shown in section 7, related to future work.

5. Some results

Following, we describe some of the most interesting results observed after the two-year period since the plan was implanted.

5.1. How to simulate some real world situations

One of the classical difficulties teaching Software Engineering is to transmit students the problems and situations that take place in real projects. It is very difficult to reproduce such situations in the classroom. On the other hand, one of the stages of development essential for the success of the project is the acquisition and analysis of requirements. In this context, we have done an experiment consisting on providing the students of the course *Management Software Engineering* with a requirement specification document prepared by students of the course *Software Engineering II* instead of by the course professors. Of course, the document has inconsistencies, contradictions, lack of precision and almost all those deficiencies that a real project document would have. Students must be able to solve those problems with the professor on the role of user. Afterwards they have to do an analysis of the application. The experience has been very positive because:

- It allowed students to face a specification document with deficiencies, which will be customary in their professional experience.
- It encouraged a higher interaction professor-student, by having the professor in the role of final user.
- Students used a specification that conforms to a standard (IEEE 830-1998) that, even if it is not used in their future professional experience, provides a model with a lot of similarities.

5.2. Selected tools

The tools used for the realization of practical work are Rational™ products. Specifically, students used Rational RequisitePro for the requirement capture and Rational Rose for the use case modeling, analysis and design of the application. Despite the fact that for modeling there are some free distribution tools like ArgoUML, we consider more likely that in the professional environment students find tools like Rational. The main disadvantage is the high price of these tools. Even with the 80% discount for Universities, to equip a laboratory for 20 users with Rose and RequisitePro costs about 33245 euros.

The experience with this tool has been positive. Students have been able to experiment the facilities of a very powerful CASE tool and to see the effort it saves. However, we have also

seen that they use very few facilities of the tool, just the ones required to do the practical work. We must find ways to encourage students curiosity in this respect.

5.3. SE in the second year

Everybody knows that learning SE is a difficult task, not only because of required previous knowledge, but because of the abstraction level necessary. In the TESC degree, this discipline is taught in the second year, having only 4.5 mandatory credits assigned (see Figure 2). In the second year, simultaneously to SE, students take the course *Data and Information Structure* and they have not taken yet any courses on databases. This implies that students have not developed yet an adequate ability for abstraction, essential for the analysis and design stages. On the other hand, they do not have any notions about data repositories (important for modeling and implementation). To this, we have to add that this course is the only mandatory one of SE in the degree, and because of that, it is the only opportunity that the professor has to provide the student with a minimum set of knowledge about SE.

To solve these problems, and given that the study plan cannot be modified, this course focuses on analysis and design, without considering management aspects proper of more advanced courses. These topics are treated in an applied way, taking into account the existing restriction of practical credits. Besides, the object oriented paradigm has been selected because its knowledge is considered essential nowadays for a Computer Science degree.

6. Conclusions

A plan for teaching SE in two levels has been developed:

- The first one covers the most important aspects of software development and some management ones.
- The second one extends the previous concepts and presents the student more advanced topics of SE, such as software architecture, reengineering or components. It also extends the knowledge in management and covers some topics not seen in the first level, like software quality.

In this manner, we achieve a continuous line of learning in two levels, in such a way that students finishing the first level have the basic knowledge of the SE discipline and may start developing software by applying some engineering principles. On the other hand, students finishing the second level have a more solid knowledge of the basic principles and extend their knowledge on the development stages, strengthening the concepts already acquired in the first level. Furthermore, these students come out prepared for an essential task in any engineering: the planning and management of projects. Students finishing the second level have also acquired some perspective on the newest advances in the area, which may attract their interest towards a wider training.

7. Future work

As we already said, the first goal in the short term is the implantation of the whole plan described, that will take place in the academic year 2003-2004. In parallel with this, we have to design the contents of the SE courses that will be taught in the CE degree that will start in the academic year 2002-2003. We also have to work on the next renewal of the study plan for the TESM and TESC degrees. All this will happen in the framework of a near review of the European university structure according to the Bologna Declaration and the Bricall report [1].

In the meanwhile, we have to concentrate on those aspects open to improvement. On the one hand, we may attack one of the major problems that the teaching of SE has, that is, the time restriction (in credits) of the courses. This restriction prevents the development of software projects with an appropriate size, that would show the true power of this engineering. We think that a possible solution (already implanted in other universities) is the realization of a medium size project along several academic years. We know the efforts of planning and coordination that this idea entails, but we think that this practice will notably improve the quality of SE teaching. On the other hand, we think that we may improve the interrelations among the courses that configure this discipline. In section 5 we indicate the relationships currently existing among these courses. However, these relationships will disappear with the start of the new stages of the proposed plan, but new relationships will appear. An example is the course *Software Evolution* that may takes its practical input from the practical work done in the course *Software Engineering I*.

With respect to the tools to use for the practical work, we will continue making an effort to get the necessary equipment in the laboratories. We have already asked for the licenses for Rational Robot, Test Manager and PurifyPlus to be able to undertake the testing activities. Depending on the budget of the Rey Juan Carlos University and the policies of Rational Rose for university centers, we will try to equip a laboratory with all the tools necessary to support the software development and maintenance activities.

Finally, but not less importantly, we are starting to study the future integration of the disciplines SE and Databases (DB). Nowadays it is almost impossible to find a software product that does not have a database associated, or in more general terms, a data repository. The integration of both disciplines is a fact in industrial environments and we think it must be in the academic environment. Setting the inter-relations among the courses of SE and DB should drive us to achieve this goal.

8. References

[1] Bricall, J.M. (2000). *University Report 2000*. Retrieved on April 3, 2002 from http://www.campus-oei.org/oeivirt/bricall.htm

[2] Computing Curricula (December 2001). *Computer Curricula - Computer Science Volume.* (Final Report). The Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery.

[3] Computing Curricula (August 2002). *Computer Curricula – Software Engineering Volume*. (First Report). The Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery.

[4] University Council (1993). *New studies for the computer engineer, the technical engineer in software management and the technical engineer in systems computing* [Professional Monographs]. Universidad-Empresa Foundation and University Council.

[5] SWEBOK (May 2001). *Guide to the Software Engineering Body of Knowledge*, trial version (Version 1.00). Software Engineering Coordinating Committee.